TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

Tutkintotyö

Petteri Saario

LAADUNVARMISTUS MONEN TOIMITTAJAN OHJELMISTOPROJEKTISSA

Työn ohjaaja lehtori Erkki Hietalahti
Työn teettäjä Digia Oyj, valvojana diplomi-insinööri Vesa Poikajärvi
Tampere 2008

TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka
Petteri Saario
Laadunvarmistus monen toimittajan ohjelmistoprojektissa
Tutkintotyö 33 sivua + 3 liitesivua
Työn ohjaaja lehtori Erkki Hietalahti
Työn teettäjä Digia Oyj
Elokuu 2008
Hakusanat laatu, laadunohjaus, laadunvarmistus

TIIVISTELMÄ

Työn tavoitteena oli kertoa laadunvarmistukseen kehitetystä ohjelmasta nimeltä Portinvartija. Toisena tavoitteena oli selvittää erilaisten laadunohjaukseen tarkoitettujen menetelmien periaatteet ja hyödyllisyys. Oman lisänsä työhön tuo monen toimittajan projektin käsite. Siinä monta yritystä tekee yhtä ohjelmaa. Tämä asettaa omat haasteensa. Se oli myös yksi syy siihen, miksi Portinvartijaa ryhdyttiin tekemään. Tarvittiin työkalu, joka varmistaa, että jokaisen yksikön tekemät ohjelmiston osat ovat laadultaan vaatimukset täyttäviä.

Työssä esitellään Perl-ohjelmointikielellä tehdyn Portinvartijan rakenne ja sen tapa tarkastaa ohjelmiston laatua. Tämän lisäksi kerrotaan erilaisista meneltelmistä ohjelmiston laadun parantamiseksi. Lisäksi ohjelman jatkokehitysajatuksina syntyivät ideat graafisesta käyttöliittymästä ja historiatietojen keruusta. Näiden tietojen avulla ohjelmiston laadun kehittymistä voisi seurata.

TAMPERE UNIVERSITY OF APPLIED SCIENCES
Computer Systems Engineering
Software Engineering
Petteri Saario
Quality assurance in software project with multiple contractors
Thesis 33 pages + 3 appendix pages
Thesis Supervisor Lecturer Erkki Hietalahti
Commissioning Company Digia plc
August 2008
Keywords quality, quality control, quality assurance

ABSTRACT

Purpose of this thesis was to give accurate information about quality assurance software, which was developed to measure the quality of specific larger program. Other purpose was to present methods for quality control. One special thing in this thesis is the concept of multiple contractors. This means that many companies work on same software project, developing program for one client. The main reason why the quality assurance software was designed was the need to be assured that every part of the program meets the quality standards. The resulting quality assurance software is presented in detail in this thesis. Also, ideas for future development are presented in the thesis. These include a graphical user interface and gathering of results. After the results are saved, they can be used to create different kinds of graphs, which show how the quality is developing.

Alkusanat

Työni on tehty Digia Oyj:n testausautomaatioprojektiin. Kiitos kaikille työn teossa avustaneille. Erityisesti haluan kiittää työni valvojaa Vesa Poikajärveä työn oikeaan suuntaan viemisestä ja motivoinnista. Kiitos myös Tommi Toropaiselle työn mahdollistamisesta. Ohjaajalleni Erkki Hietalahdelle kiitos työn ohjauksesta. Suuri kiitos Eijalle kannustamisesta ja tuesta.

Tampereella 29.8.2008

_____

Petteri Saario

SISÄLLYSLUETTELO

SANASTO

Julkaisu	Asiakkaalle toimitettava valmis versio ohjelmistosta
Savutestaus	Ohjelman perustoiminnallisuuden varmistamiseen käytetty testaus
Tuote	Fyysinen hyödyke tai palvelu, jota myydään asiakkaalle
Scrum	Iteratiivinen kehitysprosessi, jota käytetään ketterässä ohjelmistokehityksessä
Moduuli	Kokonaisuuksia, jotka keskittyvät tietyn asian tekoon ohjelmiston sisällä
XML	Merkintäkieli, jolla tiedon merkitys on kuvattavissa tiedon sekaan. Käytetään yleisesti tiedon välitys- ja tallennusformaattina.
CSV	Tiedostomuoto, jolla tallennetaan yksinkertaista taulukkodataa
Excel	Laajasti käytetty taulukkolaskentaohjelma
Perl	korkean tason dynaaminen ohjelmointikieli
Staattinen analyysi	Lähdekoodin laadun tarkastaminen automaattisesti työkaluohjelmia hyväksikäyttäen

1 JOHDANTO

Laatua pidetään entistä merkittävämpänä tekijänä minkä tahansa yrityksen menestymisessä. Laatuajattelu on siirtynyt erillisen tarkastusosaston vastuulta koko yrityksen johtavaksi periaatteeksi. Myös ohjelmistokehityksessä laadun merkitys on valtava: vuonna 1994 ohjelmistoprojekteista jäi 25 prosenttia valmistumatta ja lopuissakin kustannukset ylittyivät keskimäärin puolella /1/. Tämän vuoksi ohjelmistotuotannon laadun parantamiseen on kiinnitetty paljon huomiota. Yleisesti ottaen ollaan sitä mieltä että laadukas prosessi tuottaa laadukkaita tuotteita. Siksi on kehitetty erilaisia menetelmiä ja standardeja, joilla ohjelmistotuotannon laatua voidaan mitata ja parantaa.

Nykypäivän ohjelmistokehityksessä on normaalia, että asiakkaan ohjelmistoa kehittää moni alihankkija. Tällaisessa tilanteessa on myös työssä esimerkkinä käytetty projekti. Projektissa syntyi tarve tehdä ohjelma, jonka avulla voi varmistaa, että kaikkien yksiköiden tuottamat moduulit ovat laadultaan riittävän hyviä. Tässä työssä kerrotaan tarkemmin laadunvarmistukseen tehdystä ohjelmasta nimeltään Portinvartija. Työssä esitellään myös erilaisia laadunohjaukseen käytettyjä menetelmiä. Nämä menetelmät ovat käytännössä hyväksi havaittuja keinoja ohjelmiston laadunohjaukseen. Tarkoituksena on, että tämän työn pohjalta syntyy kuva siitä, miten eri menetelmät auttavat omalta osaltaan laadukkaan tuotteen syntymisessä.

2 LAATU KÄSITTEENÄ

Laadulla on erittäin monta määritelmää. Käsitteenä laatu on laaja ja suhteellinen. Seuraavassa pyritään avaamaan laadun käsitettä teoriatasolla ja ohjelmistotuotannon näkökulmasta.

2.1 Laadun määritelmä

Perinteisesti sanakirjoissa laatu kuvataan erinomaisuutena tai hyvien ominaisuuksien summana. Silloin keskitytään enemmän hyvän laadun määrittelyyn. Laajasti käytetty laadun määritelmä keskittyy siihen, kuinka hyvin tuote vastaa sille asetettuja vaatimuksia ja määrittelyjä. Tämä lähestymistapa tuo esille sen, että laatu on suhteellista. Jotta laatua voidaan mitata, se täytyy sitoa joihinkin ennalta määrättyihin kriteereihin. Toinen laadun määritelmä kertoo laadun olevan tuotteen sopivuus käyttötarkoitukseensa. Tällä tarkoitetaan sitä, että tärkeintä laadun määrittelyssä on loppukäyttäjän näkökulma. Parhaassa tilanteessa edellä mainitut päämäärät tarkoittavat samaa – eli määrittelyt ja vaatimukset sopivat yhteen loppukäyttäjien tarpeiden ja mielikuvien kanssa. /2/

2.2 Laadun eri näkökulmat

Laatua voidaan tarkastella monesta eri näkökulmasta. Evans ja Lindsay /3/ määrittelevät viisi eri näkökulmaa laatuun.

2.2.1 Käsitysnäkökulma

Käsitysnäkökulma vastaa perinteistä hyvän laadun eli erinomaisuuden näkökulmaa. Tuotteen laatua ei voi mitata tästä näkökulmasta tarkasti. Jotkin tuotteet vain ovat

laadukkaita, se on jotain minkä vain tietää. Esimerkiksi Rolex-kelloihin ja Mercedes-Benz-autoihin kohdistuu tällainen laadukkuuden mielikuva. Täytyy kuitenkin muistaa, että mielikuvat ja näkemykset vaihtelevat, joten myös tuotteiden laatu vaihtelee sen mukaan, kuka niitä arvioi.

2.2.2 Tuotenäkökulma

Tuotenäkulmassa lähdetään siitä, että laatu on jonkin mitattavan ominaisuuden määrä tuotteessa. Näin ollen laadusta kertoo esimerkiksi ommelten määrä senttimetrillä tai sylintereiden määrä moottorissa. Suurempi määrä tekee laadukkaamman tuotteen. Myös tästä näkökulmasta laadun mittaaminen tarkasti on vaikeaa, koska eri ihmisten arviointikriteerit ovat erilaiset.

2.2.3 Käyttäjänäkökulma

Käyttäjänäkökulma kertoo, että laadun määrittää se, mitä asiakas haluaa ja tarvitsee. Siinä arvioidaan tuotteen sopivuutta sen tarkoitettuun käyttöön. Tästä seuraa se, että myös tämä näkökulma vaihtelee, koska eri ihmisillä on eri tarpeita. Esimerkiksi Cadillac ja Jeep, jotka molemmat ovat yleisesti laadukkaina pidettyjä autoja, sopivat eri käyttötarkoituksiin ja eri käyttäjille. Ensimmäinen on laadukas suurien nopeuksien moottoritiellä, kun taas toinen on parhaimmillaan vaikeakulkuisessa maastossa.

2.2.4 Arvonäkökulma

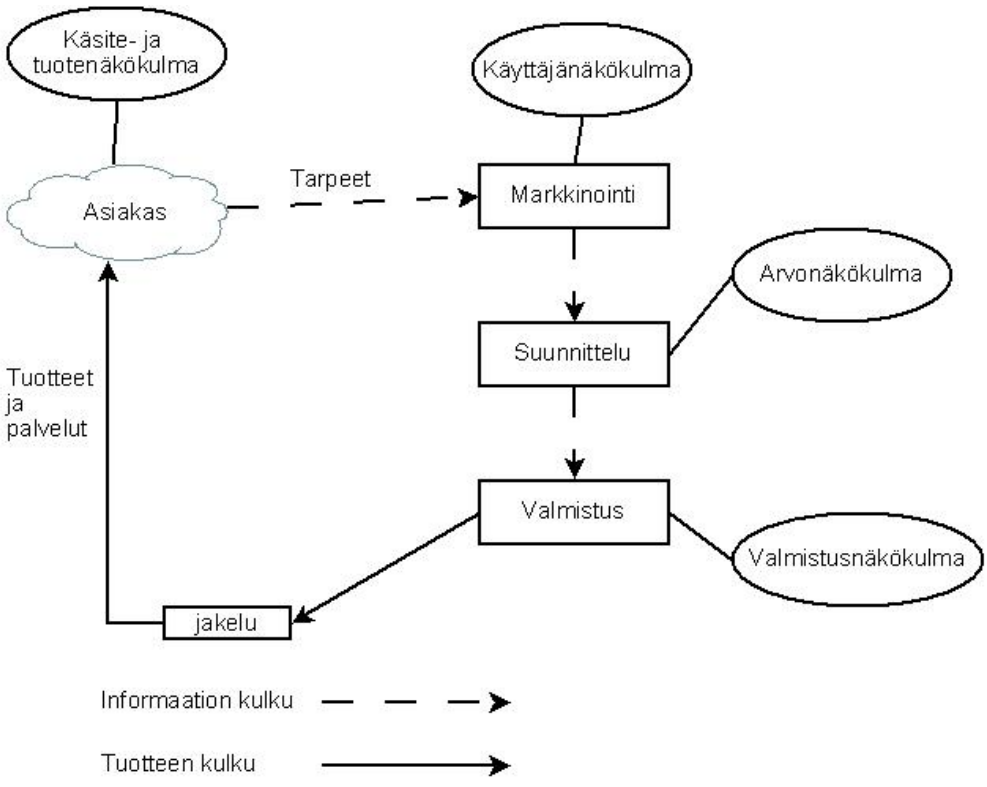
Arvonäkökulma tarkoittaa suhdetta tuotteen hankintahinnan ja siitä saatavan hyödyn välillä. Laadukas tuote voi olla sellainen, jota myydään muita vastaavia tuotteita halvemmalla tai sitä myydään samaan hintaa, mutta se on ominaisuuksiltaan parempi kuin muut vastaavat.

2.2.5 Valmistusnäkökulma

Valmistusnäkökulma keskittyy tuotteen tuotantoon. Se tarkoittaa tuotteen vastaavuutta määrittelyynsä. Ennen tuotteen tekemistä sille laaditaan määrittelyt siitä, minkälaisia ominaisuuksia tuotteeseen tulee. Koska tuotannossa ei koskaan pystytä täysin identtisiin suorituksiin, määritellään virherajat, joita enempää tuote ei saa poiketa suunnitellusta. Jos jonkin osan mitta on 10 cm ja sen virheraja on 0,5 cm, niin osa on laadukas kun sen mitta on 10±0,5 cm. Tämän näkökulman hyvä puoli on se, että laatua voidaan mitata. Täytyy kuitenkin muistaa, että määrittelyt ovat turhia, jos ne eivät vastaa käyttäjän tärkeinä pitämiä asioita.

2.2.6 Laatunäkulmat tuotantoketjun eri vaiheissa

Edellä mainitut näkökulmat auttavat ymmärtämään sitä, kuinka ihmiset tuotantoketjun eri osissa näkevät laadun. Seuraavassa kuvassa asiaa on havainnollistettu.



Kuva 1 Laadun näkökulmat tuotantoketjun eri vaiheissa /3/

Asiakas, joka tuotteen ostaa, näkee laadun lähinnä käsitys- ja tuotenäkökulmista. Yritykselle taas on tärkeää tunnistaa asiakkaan tarpeet ja tehdä tuotteita, jotka vastaavat näihin tarpeisiin. Markkinoinnin tehtävä on määrittää nämä tarpeet. Tämän vuoksi markkinointihenkilöille käyttäjänäkökulma on tärkein laadun määrittelyssä. Kun asiakkaan tarpeet ovat selvillä, niistä täytyy tehdä tuotteen määrittely. Tämä on tuotekehitys- ja suunnitteluosastojen työtä. Tärkeintä näille osastoille on tasapainottaa tuotteen ominaisuuksien suhde valmistuskustannuksiin. Tämä vastaa laadun arvonäkökulmaa. Viimeisenä tuotteen valmistajien vastuulla on tuottaa määrittelyjä vastaavia tuotteita, eli he näkevät laadun valmistusnäkökulmasta. /3/

2.3 Laadunohjaus

Laadunohjaukselle (eng. quality control) on olemassa monia määrittelyjä, aivan kuten laadulle. Ne painottavat eri asioita, mutta kaikissa on sama ydinajatus. Laadunohjaus käsittää kaikki aktiviteetit ja tekniikat, jotka tähtäävät tyydyttävän laatutason saavuttamiseen. Tyydyttävä laatutaso saavutetaan, kun kaikki tuotteelle asetetut laatuvaatimukset täyttyvät.

Laadunohjaus on kehittynyt viimeisen puolen vuosisadan aikana merkittävästi. Tässä kehityksessä on viisi eri vaihetta:

1. ammattitaito
2. esimiehen laadunohjaus
3. tarkastukset
4. tilastollinen laadunohjaus
5. totaalinen laadunohjaus.

Näistä vaiheista moni voi esiintyä samaan aikaan. Esimerkiksi jokin yritys käyttää vain tarkastuksia saavuttamaan tietyn laatutason. Käsityöläiselle oma ammattitaito on tärkein väline, jolla laatu turvataan. Tilastollinen laadunohjaus on tärkeää, koska se on paras tapa seurata laadun kehitystä ja löytää ongelmakohdat, joihin puuttua.

Totaalinen laadunohjaus eroaa muista siinä, että se pitää sisällään kaikki edellä olevat vaiheet ja enemmän. Se painottaa koko yhtiön sitoutuneisuutta laatuun ja jatkuvaan kehitykseen. Varsinkin johdon rooli on tärkeä. Päätöksenteon tärkein ohjenuora on tehdä tuotteita mahdollisimman vähin resurssein, mutta kuitenkin siten että ne tyydyttävät asiakkaan tarpeet.

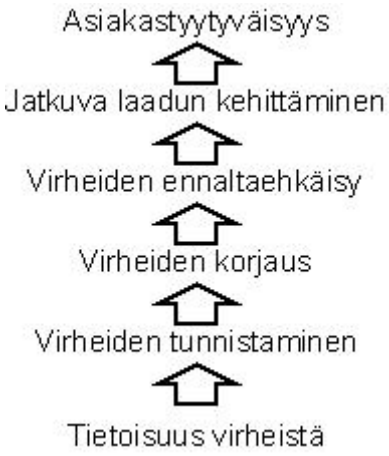
2.4 Laadunvarmistus

Laadunvarmistus (eng. quality assurance) liittyy aiemmin mainittuun laadunohjaukseen. Sillä tarkoitetaan kaikkia niitä aktiviteettejä ja tekniikoita, joilla pyritään varmistumaan siitä, että laadunohjaus tuottaa laatutason saavuttavia tuotteita. Toisin sanoen laadunvarmistus kertoo, toimiiko laadunohjaus. Laadunvarmistukseen kuuluu myös laadunohjauksen epäkohtien löytäminen ja niihin puuttuminen. /2/

2.5 Laadunhallintajärjestelmä

Laadunohjauksen kehittynein taso kuvattiin vielä vähän aikaan sitten termillä laatujärjestelmä. Tämä termi korvattiin kuitenkin ISO-9000-standardissa termillä laadunhallintajärjestelmä. Standardi määrittelee termin näin: ”Johtamisjärjestelmä, jonka avulla suunnataan ja ohjataan organisaatiota laatuun liittyvissä asioissa.” /1/ Laadunhallintajärjestelmä lähtee ylimmän johdon tasolta ja vaikuttaa kaikkiin yrityksessä oleviin työprosesseihin. Sen tärkein päämäärä on asiakastyytyväisyys. Tämän perusteella voidaan päätellä, että laadunohjaus (kohta 2.3) ja laadunvarmistus (kohta 2.4) ovat osa laadunhallintajärjestelmää.

Laadunohjauksen kehitystä kohti laadunhallintajärjestelmää voidaan tarkastella myös kehityksenä virheiden havainnoinnista niiden ehkäisyyn ja sitä kautta jatkuvaan kehitykseen, lopullisena päämääränä asiakastyytyväisyys. Tätä kehitystä on havainnollistettu kuvassa kaksi. /2/



Kuva 2 Laadunohjauksen kehitys kohti asiakastyytyväisyyttä /2/

3 MONEN TOIMITTAJAN PROJEKTI

Tässä luvussa kerrotaan, mitä tarkoitetaan monen toimittajan projektilla ja mitä ongelmia se tuo mukanaan.

3.1 Monen toimittajan määritelmä

Monen toimittajan projektissa on kyse tilanteesta, jossa useampi eri yritys tekee samaa tuotetta. Nämä yritykset voivat toimia keskenään suorassa yhteisprojektissa. Nykyajan maailmassa on kuitenkin huomattavasti yleisempää, että yritykset ulkoistavat toimintojaan ja tätä kautta tullaan tilanteeseen, jossa useampi alihankkija tekee asiakkaan ohjelmistoa. /4/

3.2 Kommunikaation vaikeus

Kun ohjelmistoprojektit levittäytyvät moniin eri maihin ja yrityksiin, kokonaisuuden hallinta vaikeutuu. Esimerkkiprojektissa kehitystyötä tehdään monella mantereella. Pääpaino on kuitenkin Suomessa, jossa ohjelmisto pidetään koossa ja vastataan kaikkien moduulien yhteensovittamisesta. Tärkeää tässä työssä on se, että kaikkien yksiköiden kehitystyö on laadultaan samantasoista. Näin ei kuitenkaan aina ole. Muiden maiden käytännöt saattavat olla löysempiä ja ongelmat ratkaistaan helpoimmalla tavalla, vaikka se saattaisi aiheuttaa ongelmia jossain muualla. Ei siis nähdä kokonaisuutta. Yleisimpiä vianaiheuttajia ovat tilanteet, joissa ei ole muistettu tai katsottu aiheelliseksi kertoa toisille yksiköille, että jokin moduulin osa nimetään uudelleen tai poistetaan käytöstä. Tämä johtaa tilanteisiin, missä huomataan, että jokin ennen toiminut asia ei yhtäkkiä toimi. Vasta vikaa selvitettäessä huomataan, että olisi pitänyt reagoida jotenkin muutokseen, joka toisessa moduulissa on tehty.

Suuri syy tiedon jakamisen ongelmiin on usein se, että alihankkijat ovat keskenään kilpailijoita. Tietoa ei haluta jakaa kilpailevaan yritykseen vähimmäismäärää enempää.

4 LAADUNOHJAUS PROJEKTISSA

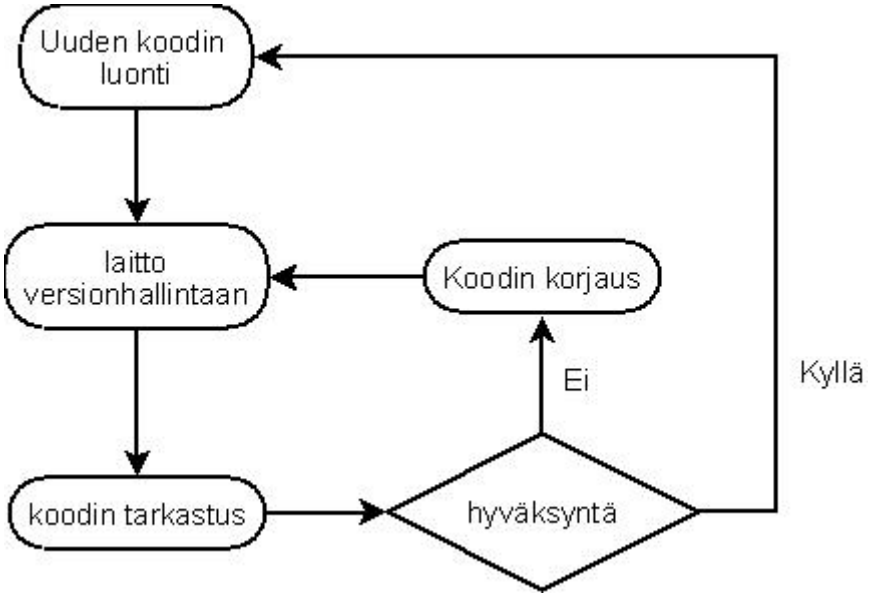
Tässä luvussa kuvataan sääntöjä ja käytäntöjä, joilla laadunohjausta suoritetaan esimerkkiprojektissa. Projekti on testaus- ja testikehitysprojekti älypuhelinjärjestelmille. Siinä käytetään ketteriä menetelmiä ja projektimallina on Scrum. Tästä syystä projektin julkaisusykli on erittäin lyhyt - kaksi viikkoa. Tämä pitää uuden sisällön melko pienenä ja takaa sen, että järjestelmä on koko ajan toimiva ja ajan tasalla.
Seuraavat käytännöt eivät suoraan nojaa mihinkään standardiin, vaan niissä on otettu hyväksi havaittuja osia eri lähteistä.

4.1 Vertaistarkastus

Vertaistarkastus on järjestelmä, joka on saanut innoituksensa avoimen lähdekoodin projektista. Järjestelmä levisi hieman muunneltuna erään yrityksen projekteihin. Järjestelmän käyttöönotosta on olemassa artikkeli, jonka pohjalta järjestelmä otettiin myös tässä projektissa käyttöön. /5/

4.1.1 Vertaistarkastuksen toimintaperiaate

Lähtökohta vertaistarkastuksessa on se, että kehittäjät tarkastavat toistensa koodia usein, jotta tarkastettavat kokonaisuudet ovat pieniä ja tarkastus nopeaa. Muutenkin tarkastus integroidaan työhön mahdollisimman hyvin ja asetetaan etusijalle. Käytännössä tämä tarkoittaa sitä, että toisen koodit on tarkastettava ennen kuin alkaa itse kirjoittaa koodia. Seuraavana oleva kuva havainnollistaa järjestelmää.



Kuva 3 Vertaistarkastuksen toimintaperiaate

Toiminta lähtee liikkeelle uuden koodin luomisesta tai koodimuutosten teosta. Versionhallintaan koodi laitetaan mielellään joka päivä, jotta muutokset saadaan pidettyä pieninä. Versionhallintaan ei kuitenkaan saa laittaa koodia, joka ei käänny. Joka yö ajettava automaattinen ohjelma listaa kaikki muutokset eri moduuleihin ja lähettää sähköpostilla yksittäisten moduulien muutokset tarkastettavaksi postituslistalta järjestyksessä valituille henkilöille, yksi moduuli per henkilö. Samat viestit lähtevät myös moduuleihin nimetyille vastuuhenkilöille. Tarkastajaksi valitun henkilön täytyy tarkastaa koodi viivyttelemättä. Kun tämä on tehty, tarkastaja lähettää viestin, jossa hän kertoo, hyväksyykö hän tehdyn koodin. Jos ei, tekijä on velvollinen korjaamaan koodin ensi tilassa. Koodia kierrätetään tarvittaessa monta kertaa saman prosessin läpi, kunnes tarkastaja sen hyväksyy. Tekijän pitää myös kuitata tarkastajan viestit saaduksi joka kerta. Näin järjestelmä pysyy ajan tasalla siitä, että tarkastajan viestiin on reagoitu ainakin jotenkin.

4.1.2 Vertaistarkastuksen hyvät ja huonot puolet

Vertaistarkastuksella saadaan koodin tarkastus osaksi jokapäiväistä työtä. Kun muutokset pysyvät pieninä, niihin on helppo reagoida ja tarkastus on nopeaa. Nopeuteen vaikuttaa paljon myös prosessin automatisointi. Järjestelmä huolehtii

siitä, että kaikki muutokset tulevat jonkun tarkastettaviksi. Se myös pitää kirjaa prosessin kulusta. Kun koodia tarkastetaan usein, virheet löytyvät nopeasti ja niiden korjaaminen on helppoa. Juuri tehty koodi on tuoreessa muistissa, toisin kuin perinteisissä projekteissa, joissa tarkastetaan suuria koodimääriä kerralla ja varsin pitkän ajan kuluttua koodin tekemisestä. Järjestelmä myös kasvattaa osaamista, kun nuoremmat kehittäjät saavat koodiinsa kommentteja kokeneilta projektin jäseniltä.

Vertaistarkastuksen toimivuus lepää täysin työntekijöiden harteilla. Vaikka järjestelmä huolehtii siitä, että tarkastukset tulevat jonkun vastuulle, mikään ei estä kuittaamasta viestejä koodia oikeasti tarkastamatta. Onkin tärkeää, että ihmiset sisäistävät järjestelmän osaksi arkea. Tämä tarkoittaa myös sitä, että koodi tulisi laittaa versionhallintaan mahdollisimman usein, eikä suinkaan kahden viikon välein, jolloin muutoksia on tuhansien rivien edestä. On myös jokaisen kehittäjän vastuulla reagoida tarkastajan kommentteihin ja laittaa koodi kuntoon. Oman vaikeutensa tuo se, että projektissamme käytetään monia ohjelmointikieliä. Tämä voi johtaa tilanteeseen, jossa tarkastajaksi joutuu henkilö, joka ei kyseistä kieltä osaa. Tämä ongelma oli aiemmin ratkaistuna kiinteillä moduulien tarkastusvastuullisilla. Ratkaisun kääntöpuolena joillekin henkilöille tuli paljon muutoksia tarkastettavaksi, kun taas toiset eivät juuri joutuneet tarkastajiksi. Tarkastaminen söi liikaa muutaman kehittäjän resursseja ja laski samalla heidän motivaatiotaan tehdä tarkastuksia.

4.2 Tarkastukset

Tarkastaminen on perinteinen tapa tutkia tehdyn työn jälkeä ohjelmistoprojekteissa. Tarkastusten perinteinen käyttö ohjelmiston elinkaarimallin mukaisten vaihetuotteiden virheiden löytämisessä ei toimi tässä projektissa, mikä johtuu ketterästä projektimallista ja lyhyestä julkaisusyklistä. Tarkastuksia käytetään isompien muutosten tai kokonaan uusien moduulien yhteydessä määrittely- ja suunnitteludokumenttien tarkastamiseen. Koodia ei läpikäydä tarkastuksissa, koska käytössä on edellä (kohta 4.1) kuvattu vertaistarkastus.

4.2.1 Tarkastuksen toimintaperiaate

Tarkastus on kokous, jonka avulla pyritään löytämään virheet käymällä läpi tarkastettavaa dokumenttia. Tarkastuksiin osallistuvat ihmiset valitsee tarkastettavan dokumentin tekijä, mahdollisesti esimiehen avustuksella. Tärkeää on kutsua vain niin paljon ihmisiä, kuin todella tarvitaan. Osallistuvassa ryhmässä on erityisrooleja: puheenjohtaja, sihteeri, tekijä ja alustaja. Myös erityisasiantuntijoita tai asiakkaan edustajia voi olla läsnä. Tekijä toimii normaalisti myös sihteerinä ja alustajana. Ennen varsinaista tilaisuutta voi tekijä esittellä tarkastettavan materiaalin. Tarkastustilaisuudessa dokumentti käydään läpi ja kommentit kirjataan muistiin. Tilaisuuden lopuksi dokumentti joko hyväksytään tai hylätään. Hyväksyminen voi tapahtua korjauksin, jolloin tekijä korjaa dokumentin ja hyväksyttää sen uudelleen puheenjohtajalla. Jos dokumentti hylätään, sovitaan heti aika uusintatarkastukselle. /1/

4.2.2 Tarkastuksen hyvät ja huonot puolet

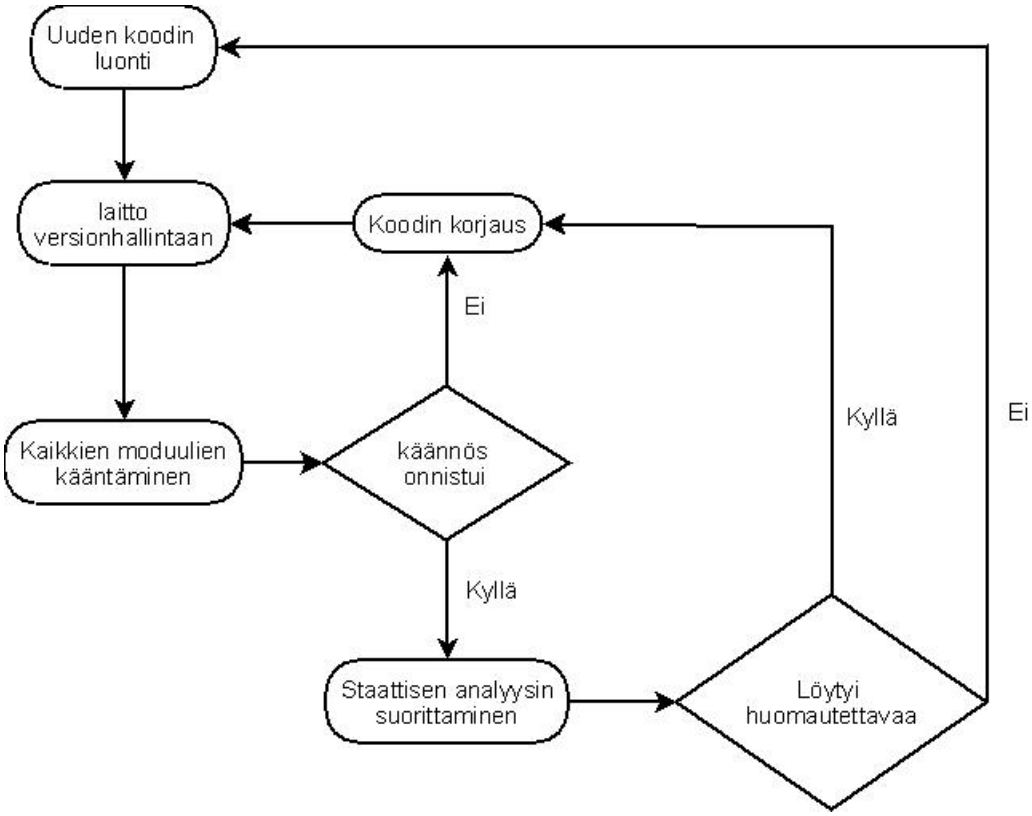
Tarkastukset ovat hyvä tapa löytää paljon virheitä. Myös virheiden kustannukset jäävät pieniksi, kun ne löydetään mahdollisimman aikaisessa vaiheessa. Tarkastuksia voidaan tehdä sellaiselle materiaalille, jota ei voida testata. Tavallisimmat ongelmat tulevat organisoinnissa. Projektin avainhenkilöille voi tulla liian suuri tarkastuskuorma. Tällöin tarkastukset voivat viivyttää projektin etenemistä. Materiaali voi myös olla keskeneräistä tai sitä voi olla liikaa. Tällöin käy helposti niin, että tarkastaminen koetaan turhauttavaksi ajanhukaksi. /1/

4.3 Jokaöiset käännökset

Jokaöiset käännökset on automaattinen järjestelmä, joka kääntää kaikki versionhallinnasta löytyvät uusimmat koodit ja ajaa niille staattisen analyysin työkaluja. Järjestelmällä pyritään takaamaan se, että versionhallinnasta löytyvä koodi on aina käännöskelpoista.

4.3.1 Jokaöisten käännösten toimintaperiaate

Kuten jo vertaistarkastusten yhteydessä mainittiin, on tärkeää, että muutokset laitetaan mahdollisimman usein versionhallintaan. Kuitenkin tulisi varmistaa, että sinne menee vain koodia, joka kääntyy. Automaattinen työkalu lataa joka yö versionhallinnasta uusimmat koodit ja yrittää kääntää ne jokaisen tuetun alustan päällä. Jos käännös menee läpi, koodeille ajetaan staattista analyysia suorittavia työkaluja. Käännösten epäonnistumisesta ja staattisen analyysin tuottamista löydöksistä lähetetään automaattisesti sähköpostia moduulien vastuullisille. Kuvassa 4 havainnollistetaan toimintaa.



Kuva 4 Jokaöisten käännösten toimintaperiaate

4.3.2 Jokaöisten käännösten hyvät ja huonot puolet

Jokaöisillä käännöksillä varmistetaan, että versionhallinnasta löytyy koko ajan käännettävissä oleva ohjelmisto. Samalla saadaan selville, ovatko moduulit

perustoteutukseltaan yhteensopivia. Staattisella analyysillä selvitetään myös koodin laatua. Jos huomautettavaa on, korjaukset koodin voi tehdä seuraavana päivänä koodin ollessa vielä tuoreessa muistissa. Ongelmia voi aiheuttaa järjestelmän luotettavuus. Käytäntö on osoittanut, että jos järjestelmään tulee vika ja se lähettää sähköpostiviestejä turhaan, ohjelmoijille tulee helposti järjestelmästä epäluotettava kuva. Silloin ajatellaan, että viestien tullessa omassa moduulissa ei oikeasti ole vikaa, vaan järjestelmä on hajalla. Tämä johtaa siihen, että myös aiheelliset varoitukset ohitetaan ilman toimenpiteitä.

4.4 Savutestaus

Savutestauksen (eng. smoke testing) tarkoituksena on testata järjestelmän perustoiminnallisuus. Testaus suoritetaan uudelle julkaisulle ennen mitään muuta testausta. Jos julkaisu ei läpäise savutestausta, se on liian epävakaa muuhun testaukseen. Tällöin julkaisu täytyy korjata ja ajaa savutestit uudelleen. Vasta hyväksytyn testiajon jälkeen julkaisua voidaan testata muuten. /6/

4.4.1 Savutestauksen toimintaperiaate

Savutestauksessa käytetään ennalta sovittua ja samanlaisena pysyvää testisettiä. Aina syklin lopussa tulevalle julkaisulle ajetaan savutestit jokaisella tuetulla alustalla. Savutestaus on pyritty automatisoimaan mahdollisimman pitkälle, jotta siitä saadaan helppo ajaa. Jos savutestauksessa ilmenee virheitä, niihin tulee reagoida välittömästi ja selvittää syyt. Vasta kun virheet on korjattu tai niille on esitetty jokin pätevä syy, voidaan todeta julkaisun olevan kunnossa.

4.4.2 Savutestauksen hyvät ja huonot puolet

Savutestaus on erittäin hyvä järjestelmä. Pitkälle automatisoituna sillä saa helposti varmuuden järjestelmän perustoiminnallisuudesta. On paljon tehokkaampaa ajaa nopea savutesti järjestelmälle ja todeta toimivuus, kuin huomata vasta oikean testauksen yhteydessä, että jokin järjestelmän perustoiminto on epäkunnossa ja aiheuttaa virheitä muissa toiminnoissa. Tässä vaiheessa virhelähteen löytäminen on huomattavasti työläämpää ja kuluttaa paljon enemmän resursseja.
Savutestauksen haaste tulee oikeiden testitapausten valinnasta. Testin pitää kattaa järjestelmän perustoiminnot ja olla nopea ja mahdollisimman automaattinen suorittaa.

5 OHJELMA LAADUNVARMISTUKSEEN: PORTINVARTIJA

Tässä luvussa kerrotaan projektissa tehdystä Portinvartija-ohjelmasta. Se vastaa projektissa tehdyn tuotteen laadun varmistamisesta. Tämä tehtävä korostuu, koska kyseessä on monen toimittajan projekti. Ohjelman tarkastettavaksi tulevat myös muiden yksiköiden tekemät moduulit. Luvussa kerrotaan Portinvartijan toimintaperiaatteesta, sen suunnittelusta ja toteutuksesta.
Portinvartija on suunniteltu tarkastamaan projektissa tehtävä julkaisu. Tiheästä kahden viikon julkaisusyklistä johtuen ohjelma on käytössä melko usein.

5.1 Yleiskuvaus

Portinvartija on uuden julkaisun laatua monin eri tavoin tarkastava ohjelma. Yleisellä tasolla sen toiminnallisuus koostuu erilaisten tarkastusten tekemisestä julkaisulle ja mahdollisten epäkohtien raportoinnista eteenpäin. Yksi julkaisu koostuu vaihtelevasta määrästä moduuleita. Ohjelma on komentorivipohjainen. Se on suunniteltu helposti laajennettavaksi, mikä asettaa omat vaatimuksensa rajapinnoille, joiden avulla oliot käyttävät toisiaan. Suunnittelussa on otettu huomioon se, että käyttäjä voi halutessaan kytkeä pois päältä ohjelman eri osia ja näin suorittaa vain halutut osat.

5.2 Ohjelmointikielen valinta

Ohjelma tekokieli on Perl. Se on korkean tason dynaaminen ohjelmointikieli, jonka kehitti vuonna 1987 Larry Wall. Perlissä on vaikutteita kielistä, kuten C, sed, awk ja sh. Syntaksi muistuttaa lähinnä C-kieltä. Seuraavana esimerkki Perlillä tehdystä ”hello world”-ohjelmasta:

```
# tämä on kommentti
print "Hello, world";
```

Perl on ensisijaisesti tarkoitettu tekstitiedostojen lukemiseen, tekstin muokkaukseen ja erilaisten raporttien luomiseen tekstin pohjalta. Tämä kieli valittiin, koska ohjelmassa tehdään paljon tekstin muokkausta ja vertailua. Perl on helppokäyttöinen ja tähtää ohjelmoijan vaivan minimointiin, toisin kuin esimerkiksi C, joka tähtää enemmän laskentatehon kulutuksen minimointiin. /7/

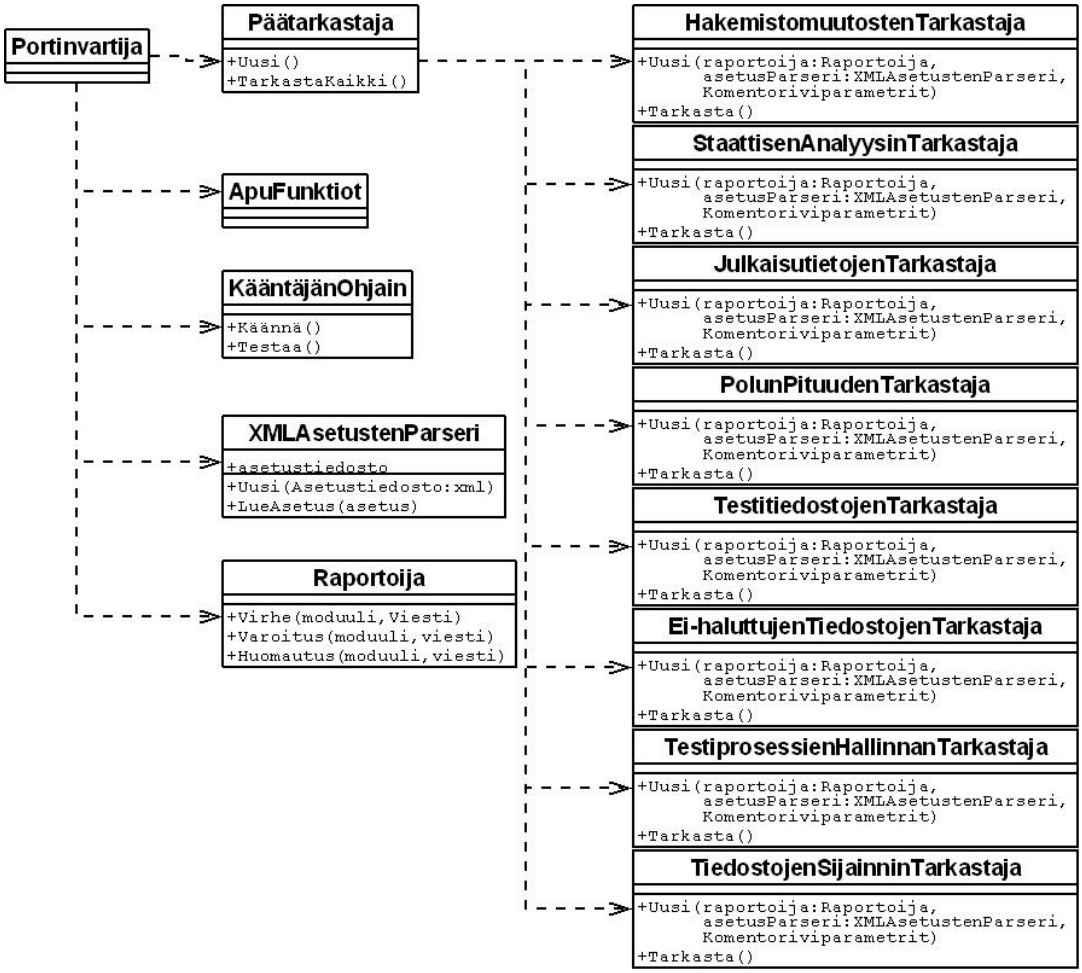
Perlin eräs vahvimmista piirteistä on säännöllisten lausekkeiden (eng. regular expressions) käyttö. Tyypillisintä on tarkastella niillä, esiintyykö jokin merkkijono suuremmassa merkkijonossa. Peruskäyttö on Perlissä erittäin helppoa, siitä seuraavana esimerkki:

```
print "osuma\n" if "Hello World" =~ /World/;
```

Esimerkkikoodi tulostaa sanan ”osuma” ja rivinvaihdon näytölle, jos sana ”World” löytyy merkkijonosta ”Hello World”.

5.3 Ohjelman rakenne

Ohjelman rakenne on lähtökohdiltaan olio-ohjelmoinnin mukainen. Ohjelmassa on luokkia toteuttamaan eri tehtäviä. Siitä löytyy myös apufunktioita, joita ei ole laitettu luokkaan, vaan ne sijaitsevat yhdessä Perl-moduulissa. Pääohjelma käyttää luokista luotuja olioita osoittimen kautta ja apufunktioita suoraan.



Kuva 5 Portinvartijan luokkakaavio

Ohjelman ydin on pääohjelma nimeltään `Portinvartija`. Tämä ohjelma luo `Päätarkastaja`-, `Raportoija`-, `KääntäjänOhjain`- ja `XMLAsetustenParseri`-luokkien instanssit ja käyttää niitä.

`Päätarkastaja` on luokka, joka luo tarkastajat kutsumalla niiden `Uusi()`-funktiota ja käyttää niiden `Tarkasta()` -funktiota ohjelmalle annetun julkaisun tarkastamiseen. Tarkastajat tekevät jokainen vuorollaan omat tarkastuksensa niille annetulle moduulille. Tarkastajien suorittamista toimenpiteistä kerrotaan lisää kohdassa 5.3.1. Laajennettavuuden takia jokaisen tarkastajan tulee sisältää funktiot `Uusi()` ja `Tarkasta()`. Tarkastajat saavat parametreinä luonnin yhteydessä osoittimet `Raportoija`- ja `XMLAsetustenParseri`-luokkien instansseihin. Lisäksi ne saavat komentoriviparametrit, jotka ohjelmalle on annettu käynnistyksen yhteydessä.

`XMLAsetustenParseri`-luokka tarjoaa muille luokille helpon tavan saada asetusten arvoja asetustiedostosta. Se hoitaa sisäisesti XML-muotoisen tiedoston parsimisen, kunhan sille annetaan haluttu tiedosto luonnin yhteydessä. `Raportoija` -luokka sisältää kolme eri funktiota, jotka tarjoavat muille luokille helpon tavan ilmoittaa havainnoista. `KääntäjäOhjain` on tarkoitettu ulkoisen järjestelmän ohjaukseen ja tulosten hakuun. Tätä ulkoista järjestelmää käytetään kääntämään julkaisussa oleva koodi ja ajamaan sille testejä automaattisesti. `Apufunktiot` on Perl-moduuli, johon on koottu erilaisia hyödyllisiä funktioita muiden osien käytettäväksi.

5.3.1 Tarkastajat

Tarkastajia ohjelman tämänhetkisessä versiossa on kahdeksan. Tässä yhteydessä tarkastajiin ei lueta päätarkastajaa, joka ei varsinaisesti tarkasta mitään. Tarkastajien asetustiedoston sisältämät asetukset löytyvät liitteestä 1. Seuraavassa tarkastajat käydään läpi ja selvitetään niiden toiminnallisuus.

5.3.1.1 Hakemistomuutosten tarkastaja

Hakemistomuutosten tarkastaja vertailee edellisen julkaisun hakemistorakennetta uuden julkaisun hakemistorakenteeseen. Se käy läpi kaikki hakemistot ja tiedostot ja ilmoittaa uudesta julkaisusta hävinneet tiedostot ja hakemistot. Tällä varmistetaan, että uudesta julkaisusta ei puutu mitään, minkä kuuluisi olla siellä.

5.3.1.2 Staattisen analyysin tarkastaja

Staattisen analyysin tarkastaja tekee staattista analyysia julkaisulle käyttäen hyväksi ulkopuolisia työkaluja. Käytännössä tarkastaja ajaa määritellyt työkalut julkaisulle ja poimii niiden tuloksista oleelliset asiat ja kirjaa ne raporttiin.

Suunnittelussa on otettu huomioon mahdollisuus lisätä käytettävien työkalujen määrää jälkikäteen. Tällä hetkellä käytettävät työkalut ovat Symscan, sen mukana tuleva Leavescan sekä CodeScanner. Koodin tarkastaminen staattisilla analysaattoreilla tuo esiin monia puutteita koodauskäytännöissä.

5.3.1.3 Julkaisutietojen tarkastaja

Julkaisutietojen tarkastaja varmistaa, että jokaisen moduulin julkaisutiedot on päivitetty. Julkaisutietojen pitäisi löytyä wiki-sivustolta jokaisen moduulin omalta sivulta. Työkalu tarkastaa, että julkaisutietojen viikkonumero vastaa uusimman julkaisun viikkonumeroa. On erittäin tärkeää, että julkaisutiedot ovat ajan tasalla, koska niistä selviää, mikä ohjelmassa on muuttunut.

5.3.1.4 Polun pituuden tarkastaja

Polun pituuden tarkastaja varmistaa, että hakemiston pituus ei ylitä määriteltyä merkkimäärää. Tämän tarkastaminen on tärkeää, koska jotkin järjestelmät eivät osaa käsitellä liian pitkiä hakemistonimiä. Polulla tarkoitetaan koko merkkijonoa, johon kuuluu hakemisto, josta tiedosto löytyy, sekä tiedoston nimi, esimerkiksi ”ohjelmisto\moduuli1\hakemisto\toinen\tiedosto.txt”. Pituus lasketaan ohjelmiston päähakemistosta eteenpäin. Suurin mahdollinen polun pituus annetaan asetustiedostossa.

5.3.1.5 Testitiedostojen tarkastaja

Ohjelmistossa on yksi erittäin tärkeä ryhmä tiedostoja. Nämä tiedostot sisältävät ohjelmiston kaikki testitapaukset, ja ne ovat se paikka johon suurin osa muutoksista kohdistuu. Ilman näitä tiedostoja testien ajaminen ei onnistu. Tiedostot ovat

listattuna erillisessä tiedostossa. Testitiedostojen tarkastaja tarkastaa, että kaikki listalta löytyvät tiedostot todella ovat julkaisussa mukana ja oikeassa hakemistossa.

5.3.1.6 Ei-haluttujen tiedostojen tarkastaja

Ei-haluttujen tiedostojen tarkastaja tarkastaa, että valmiissa julkaisussa ei ole mukana sinne kuulumattomia tiedostoja. Tällaisia tiedostoja voivat olla esimerkiksi versionhallinnan käyttämät aputiedostot ja sisäiseen testaukseen tarkoitetut tiedostot. Tarkastaja etsii näitä tiedostoja ja raportoi niiden löytöpaikat. Asetuksissa voi vapaasti määrittää, mitkä tiedostot tai tiedostotyypit ovat ei-haluttuja.

5.3.1.7 Testiprosessien hallinnan tarkastaja

Projektissa on käytössä testiprosessien hallintajärjestelmä, josta kaikki testitapaukset löytyvät. Testiprosessien hallinnan tarkastaja käy hallintojärjestelmässä olevat testitapaukset läpi ja tarkastaa, että asetuksissa määritellyt tiedot ovat oikein. Se tarkastaa myös, että julkaisussa toimitettavat testitapaukset löytyvät järjestelmästä.

5.3.1.8 Tiedostojen sijainnin tarkastaja

Tiedostojen sijainnin tarkastaja tarkastaa, että tietyt tiedostot löytyvät vain niille tarkoitetuista hakemistoista. Tällaisia tiedostoja voivat olla esimerkiksi lähdekoodi- ja otsikkotiedostot. Jos tiedostoja löytyy muista hakemistoista, ohjelma on koodauskäytäntöjen vastainen, ja on myös todennäköistä, että se ei toimi oikein. Asetuksissa tarkastajalle määritellään tietyt tiedostot tai tiedostotyypit ja hakemistot, joista niitä saa ainoastaan löytyä. Jos määritellyn kaltainen tiedosto on jossain muussa hakemistossa, se kirjataan raporttiin. Asetuksiin voi myös erikseen

määritellä tarkastuksen ulkopuolelle jätettäviä hakemistoja. Tällaiset hakemistot ovat poikkeustapauksia, jotka aiheuttaisivat turhia merkintöjä raporttiin.

5.3.2 Apufunktiot

Apufunktiot on nimensä mukaisesti kokoelma apufunktioita. Se eroaa ohjelman muista osista siten, että se on Perl-moduuli, jonka funktioita voi käyttää ilman erillisen instanssin luomista. Riittää kunhan esittelee moduulin avainsanalla `use` (esim. `use portinvartija::apufunktiot`). Apufunktiot moduuli sisältää monia yksinkertaisia funktioita, kuten ajohakemiston kyselyn.

5.4 Toteutus

Ohjelman toteutus tehtiin projektin mallin mukaan kahden viikon sykleissä. Ketterien menetelmien periaatteiden mukaisesti ohjelma oli täysin toimiva ensimmäisestä versiosta lähtien. Kehitystyön alussa vaatimukset jaettiin Scrumin mukaisesti tehtäviin. Nämä lyhyet alle 30 tunnin ohjelmointitehtävät sisälsivät jonkin ohjelman osan, kuten yhden tarkastajan, tekemisen. Toteutuksen laadunohjaukseen käytettiin vertaistarkastuksia (kohta 4.1) ja tarkastuksia (kohta 4.2).

6 JATKOKEHITYSAJATUKSIA

Tässä luvussa kerrotaan työn aikana esille tulleita jatkokehitysajatuksia Portinvartijalle.

6.1 Graafinen käyttöliittymä

Tällä hetkellä Portinvartija on täysin komentorivipohjainen työkalu. Sen käyttöä helpottaisi graafisen käyttöliittymän teko. Ohjelman käynnistyskomento on melko lyhyt, joten ohjelman parametrien antaminen graafisesti voi viedä jopa enemmän aikaa kuin komentoriviltä. Suurin hyöty saavutettaisiin siitä, että asetuksia voisi muokata graafisen käyttöliittymän kautta sen sijaan, että muokataan suoraan melko suurta XML-tiedostoa. Käyttöliittymän kautta voisi valita mitä tarkastajia käytetään ja muokata niiden parametrejä. Käyttöliittymän voisi tehdä esimerkiksi Qt-käyttöliittymäkirjastolla tai vaihtoehtoisesti Visual Studio.Net työkaluilla. Qt:n suurin hyöty olisi alustariippumaton ohjelma /8/. Tämä sopisi hienosti Portinvartijan nykyiseen toteutukseen, koska sekin on alustariippumaton.

6.2 Historiatietojen keruu

Ohjelma tuottaa monenlaisia huomautuksia julkaisun laadun epäkohdista. Tällä hetkellä nämä epäkohdat kirjataan sähköpostiviestiin, joka lähetetään määritellyille henkilöille. Koska ohjelmaa ajetaan kahden viikon välein, tietoa ohjelmiston laadusta kertyy tasaisesti. Tiedot voisi tallettaa johonkin ja käyttää erilaisten graafisten esitysten muodostamiseen. Näistä esityksistä voisi nähdä mihin suuntaan tarkastettavan ohjelmiston laatu kehittyy. Tämä palvelisi suoraan alakohdassa 2.3.1 määriteltyä tilastollista laadunohjausta. Teknisesti toteuttaminen ei olisi kovin vaikeaa. Nykyisessä toteutuksessa Raportoija kerää kaikki epäkohdat raporttiin. Lisäksi se voisi kirjata ne sopivassa muodossa tiedostoon. Esimerkiksi csv tiedosto olisi helppo toteuttaa ja Excel osaisi lukea sitä suoraan. Toinen vaihtoehto olisi

sijoittaa arvot tietokantaan ja muodostaa sieltä suoraan dynaaminen websivu, joka piirtelisi arvojen pohjalta erilaisia kuvaajia.

Epäkohtien jaotteluperiaatteena voisi käyttää tarkastajia. Vielä tarkemman esityksen saamiseksi esimerkiksi Staattisen analyysin tarkastajan tuottamia virheitä voisi jaotella tarkemmin. Tämä siksi, että kyseinen tarkastaja tuottaa eniten virheitä ja sen tuottamat virheet ovat valmiiksi jaoteltuina eri kategorioihin. Kaikki löydökset voisi lisäksi jaotella moduulikohtaisesti. Moduulien kehittäjät olisivat todennäköisesti halukkaampia puuttumaan epäkohtiin, jos löydökset ja niistä muodostuvat trendit olisivat kaikkien projektissa työskentelevien nähtävillä.

7 YHTEENVETO

Työn tavoitteena oli esitellä tarkemmin ohjelmiston laadunvarmistukseen kehitettyä ohjelmaa. Lisäksi tavoitteena oli kertoa erilaisista laadunohjaukseen käytetyistä menetelmistä. Näissä tavoitteissa onnistuin mielestäni hyvin. Ohjelma on oikeasti käytössä ja sen raporttien pohjalta on parannettu jokaisen toimittajan moduulien laatua. Portinvartijan kehitystä tullaan todennäköisesti jatkamaan jatkokehitysajatusten mukaiseen suuntaan. Työn tekeminen syvensi huomattavasti tietämystäni laadusta teoriatasolla. Myös erilaisista laadunohjausmenetelmistä kertominen syvensi omaa tietämystäni niistä ja auttoi ymmärtämään miksi jotain menetelmää käytetään ja kuinka hyödyllinen se on.

Vaikeinta työssä oli tiivistää tärkein asia erittäin laajasta laadun teoriaa käsittelevästä kirjallisuudesta. Pelkästään erilaisten laadun parantamiseen tähtäävien menetelmien analysoinnista olisi voinut tehdä yhden opinnäytetyön. Työn tekemistä hidasti myös hieman se, että suurin osa lähdemateriaalista oli englanninkielistä. Kaikille termeille ei löytynyt vakiintunutta suomennosta, vaan lähteestä riippuen sama termi tarkoitti eri asiaa.

8 LÄHDELUETTELO

Painetut lähteet

1 Haikala, I – Märijärvi, J, Ohjelmistotuotanto. Talentum. Helsinki 2004. 440 s.

2 Wadsworth, H – Stephens, K – Godfrey, A, Modern methods for quality control and improvement, toinen painos. John Wiley & Sons. Yhdysvallat 2002. 683 s.

3 Evans, J – Lindsay, W, The management and control of quality, viides painos. South-Western. Yhdysvallat 2002. 838 s.

5 Lussier, Stephane, New Tricks: How Open Source Changed the Way My Team Works. IEEE Software (vol. 21), 1/2004, s. 68-72.

6 Bach, J – Kaner, C – Pettichord, B, Lessons learned in software testing : a context-driven approach. John Wiley & Sons. Yhdysvallat 2002. 286 s.

Sähköiset lähteet

4 Ulkoistamisen parhaat käytännöt. [www-sivu]. TIEKE Tietoyhteiskunnan kehittämiskeskus ry. [viitattu 7.5.2008] Saatavissa: http://www.tieke.fi/verkostot/ict_klusteri/ict_klusterin_teemoja/ulkoistaminen/ulkoistamisen_parhaat_kaytannot/

7 perl – perldoc.perl.org. [www-sivu]. [viitattu 7.5.2008] Saatavissa: http://perldoc.perl.org/perl.html#Overview

8 Cross-Platform Application Framework - Trolltech. [www-sivu]. Trolltech ASA. [viitattu 28.5.2008] Saatavissa: http://trolltech.com/products/qt/features/platforms/desktop

9 LIITTEET

Liite 1 Portinvartijan tarkastajien asetukset asetustiedostossa

```
<inspectors>

  <inspector name=" Hakemistomuutosten tarkastaja">
    <enabled>0</enabled>
    <release-folder>c:\polku\edelliseen\versioon</release-folder>
  </inspector>

  <inspector name=" Staattisen analyysin tarkastaja">
    <enabled>0</enabled>
    <subinspector name="SymScanTarkastaja">
      <enabled>1</enabled>
      <bin>SymScan.exe</bin>
      <categories>conf\symscan_categories.csv</categories>
    </subinspector>
    <subinspector name="LeaveScanTarkastaja">
      <enabled>1</enabled>
      <bin>leavescan.exe</bin>
    </subinspector>
    <subinspector name="CodeScannerTarkastaja">
      <enabled>1</enabled>
      <bin>c:\codescanner\codescanner.exe</bin>
    </subinspector>
  </inspector>

  <inspector name=" Julkaisutietojen tarkastaja">
    <enabled>0</enabled>
  </inspector>

  <inspector name="Polun pituuden tarkastaja">
    <enabled>0</enabled>
    <max-path-length>180</max-path-length>
  </inspector>
```

```
<inspector name=" Testitiedostojen tarkastaja">
  <enabled>0</enabled>
</inspector>

<inspector name=" Ei-haluttujen tiedostojen tarkastaja">
  <enabled>0</enabled>
  <unwanted>
    <file>wrong\.exe</file>
    <file>\.svn$</file>
  </unwanted>
</inspector>

<inspector name=" Testiprosessien hallinnan tarkastaja">
  <enabled>0</enabled>
  <fields>
    <field>
      <name>Kentta07</name>
      <label>Luokka</label>
      <accepted-value>Kategoria01|Kategoria02</accepted-value>
      <severity>WARNING</severity>
    </field>
    <field>
      <name>Kentta24</name>
      <label>TestausAlusta</label>
      <accepted-value>Alusta1|Alusta2|Alusta3</accepted-value>
      <severity>WARNING</severity>
    </field>
  </fields>
</inspector>


<inspector name=" Tiedostojen sijainnin tarkastaja">
  <enabled>0</enabled>
  <folders>
    <folder>
      <name>src</name>
      <files>\.cpp$</files>
    </folder>
    <folder>
      <name>inc</name>
      <files>\.h$|\.inl$</files>
```

```
      </folder>
      <folder>
        <name>group</name>
        <files>\.mmp$|bld\.inf$</files>
      </folder>
    </folders>
    <ignore>folder1</ignore>
    <ignore>folder2</ignore>
  </inspector>
```